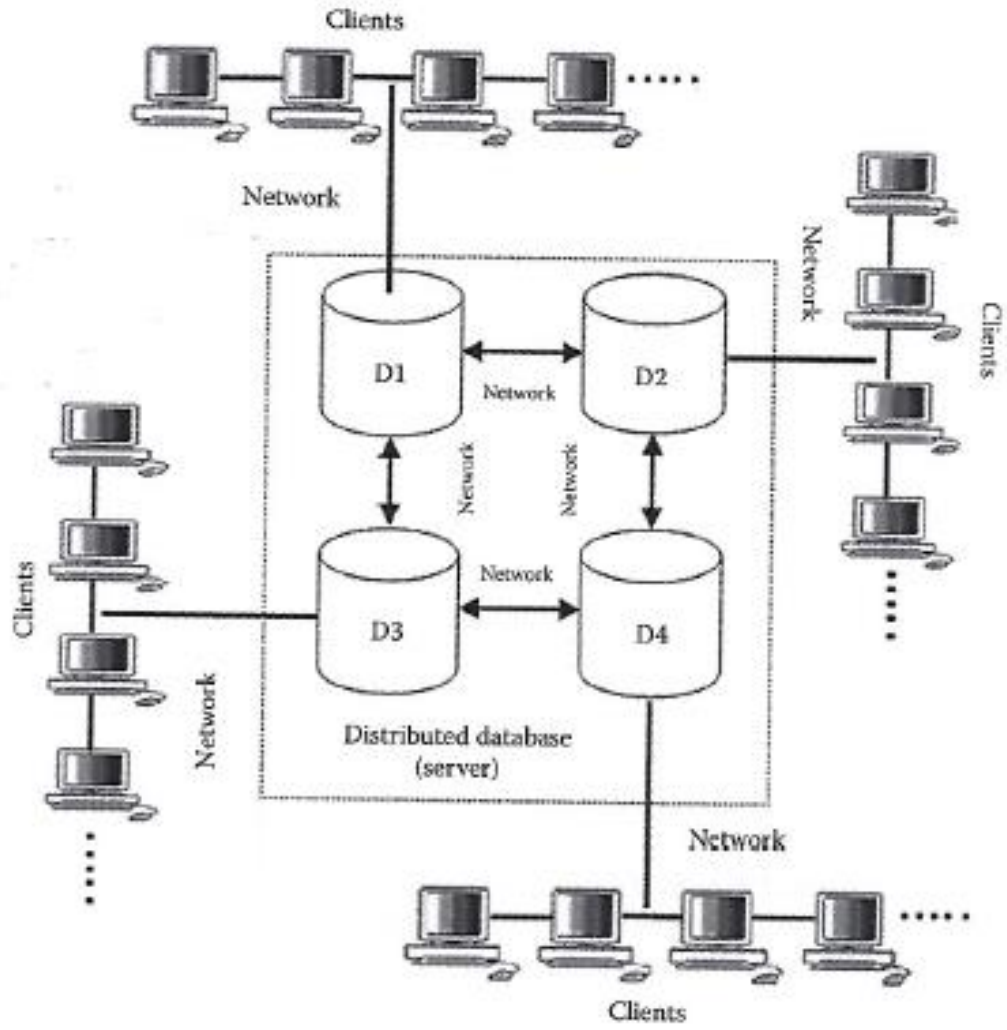
Distributed transaction processing

Chapter 1 Part 1

1. Introduction

- Expansion of huge quantity of real-time data, the dimensions of data are escalating exponentially – hard to find centralized repository that can efficiently store the data, which needs to be retrieved, manipulated , and updated using some form of management system
- DBMSs are managing DB activity as transactions, this guarantees consistency when users perform concurrent operations on them – data independence (transparency)
- However, size of data is too large to do this centralised and use computing power optimally.

- A distributed database consists of a set of interrelated databases stored on several computers distributed over a network wherein the data can be concurrently accessed and altered.
- DB server is the software that administers the database and a client is application that requests information and seek services from the server.
- Each computer is a node (client, server or both)
- Each server is managed by Its local DBMS an cooperates to preserve the consistency of the global DB.
- A DDBMS – makes distribution transparent to users. Logical DB partitioned into sections stored on one or more computers



1.1 Distributed processing and distributed DB

- Distributed processing: the use of more computer (CPU) to run an application for an individual task. Use a LAN and identify idle CPUs and parcels out programs to make use of them – uses distributed databases
- Distributed databases: data are stored across computer systems. DB system keeps track of the location of the data so the distributed character of the database is not evident to users
- Benefits of distributed processing: Resource sharing, scalability, fault tolerance/robustness, and performance/speed.

1.2 Parallel DBMS and DDMS

- Parallel processing: a subset of distributed computing where a single computer uses more than one CPU to execute programs (CPUs share information with each other).
- Distributed system: Network of independent computers that interact with each other in order to achieve a goal and do not share memory of processors. Communicate via messages over a network.
- Messages can be: to execute programs, packets of data, propel signals for behaviour

1.2 Parallel DBMS and DDMS (2)

- Parallel database management: management of data in tightly coupled multi-processor computer done by a DBMS- data are partitioned across multiple nodes.
- DDBMS: software system that authorizes the management of the distributed DB and makes the distribution transparent to the users.

1.2 Parallel DBMS and DDMS (3)

Parallel DBMS	DDBMS
Machines are physically located close to each other, for example same server room.	Machines can be located far-off from each other, for example, in diverse continent.
Machines connect with dedicated high-speed LANs and switches.	Machines can be connected using public-purpose network, (i.e. Internet)
Communication expenditure is assumed to be small	Communication expenditure and predicaments can not be ignored.
Can employ shared-memory, shared-disk or shared nothing architecture. fig1.2a,b and c	Usually employs shared-nothing architecture: fig1.2c

1.2 Parallel DBMS and DDMS (4)

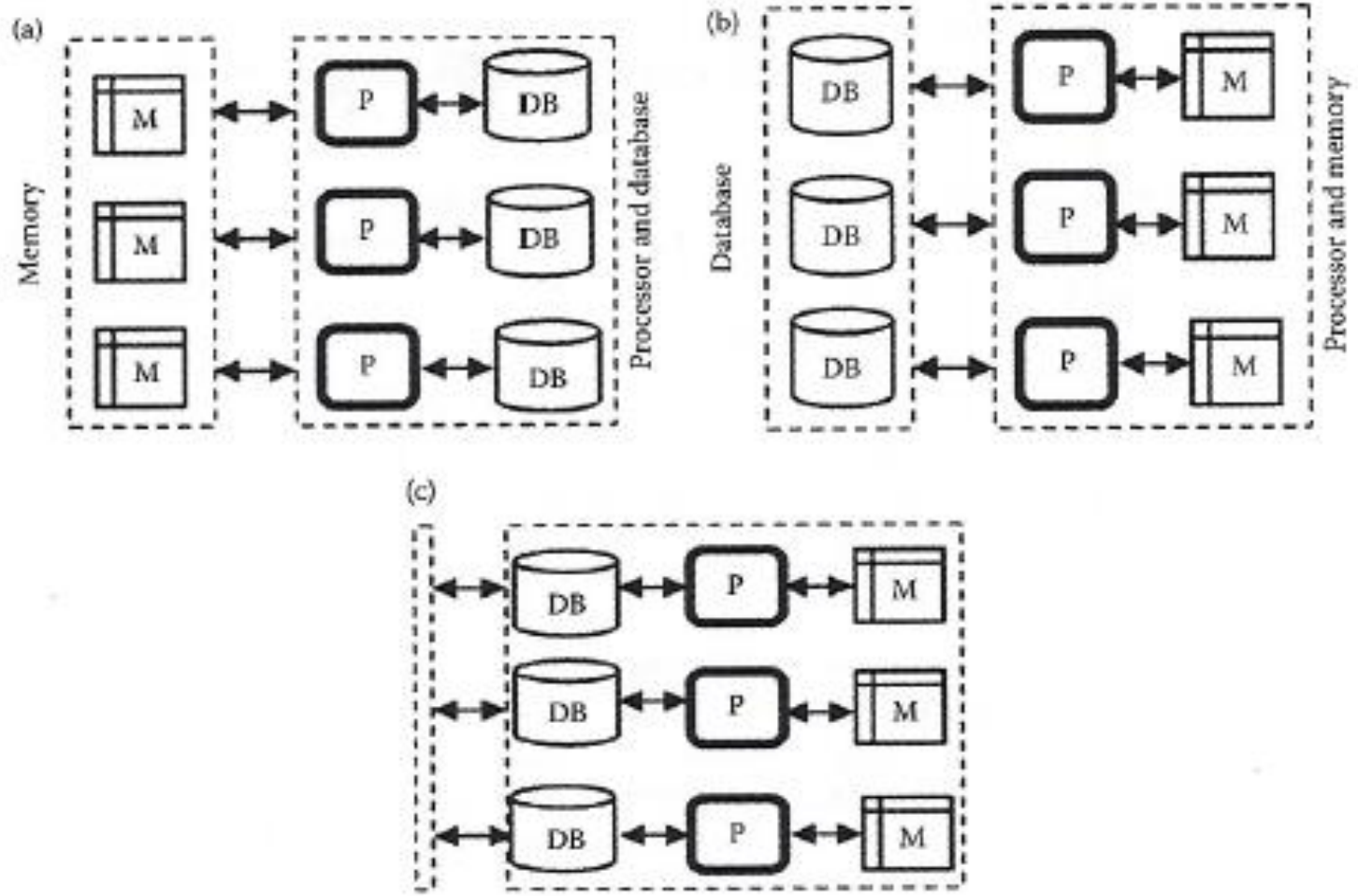


FIGURE 1.2
Different types of architecture for passing information. (a) Shared memory architecture; (b) shared disk architecture; and (c) database, processor, and memory.

1.2 Parallel DBMS and DDMS (5)

- Benefits of DDBMS:

1. Placement of data on different sites is not known to the user
2. User does not have to concern about the operational details of the processing – can issue command from any location without affecting the working of the network
3. Replication transparency – multiple copies of data at multiple sites
4. Increases reliability and availability (if one node fails, another copy of data is used)
5. Improvement in performance – keep data close to where it is needed most.
6. Scalability – expand by adding new nodes without making changes to the underlying configuration.

1.3 Distributed DB techniques

- Distributed processing has 2 main objectives:

1. To minimize irrelevant and redundant data accessed during the execution
2. To reduce data switching among nodes located at different locations.

Key is to understand how to fragment data and allocation and replication of fragments in different nodes of the distributed system.

Fragmentation: Technique of dividing relations of a DB into a number of pieces, called fragments, which are then distributed and can be sored in various computers located at different sites.

- It aims to improve reliability, performance, storage capacity, communication and security.

1.3 Distributed DB techniques (2)

Rules for fragmentation:

1. If a database D, is decomposed into fragments, D_1, D_2, …, D_n, each data item that can be found in D must in at least one fragment.
   - ✓No loss of data; consistency of distribution
2. It must be possible to define a relational operation that will reconstruct the database D from fragments, D_1, D_2, …, D_n.
   - ✓Perseverance of functional dependencies
3. If a data item X_i appears in fragment D_i, then it should not appear in any other fragment.

1.3 Distributed DB techniques (3)

- Types of fragmentation:
  - Vertical: Some of the columns are store in one computer, and the rest are stored in other computers: No selection condition is used.
  - Horizontal: Different rows are stored on different computers, depending on a condition
  - Mixed / hybrid: combination of other two

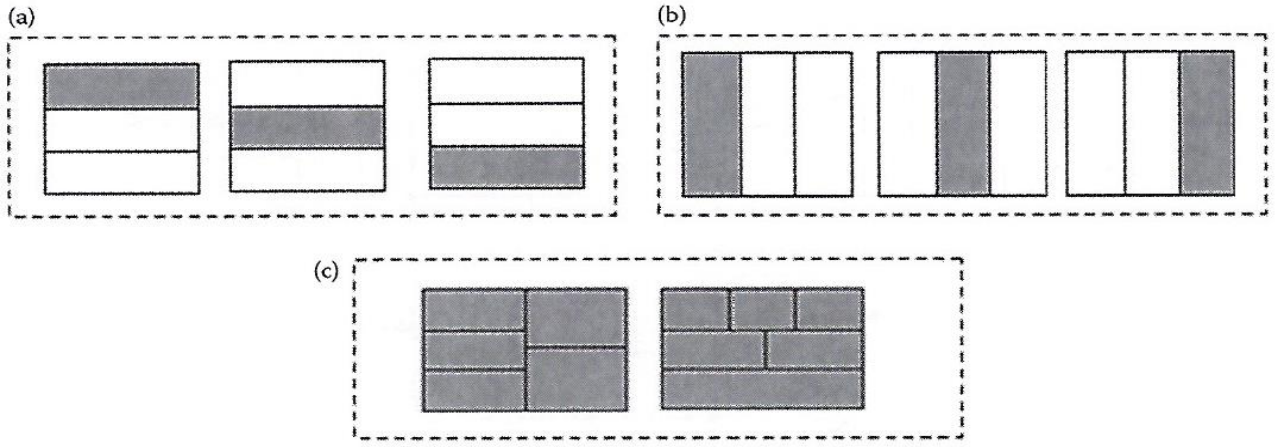


FIGURE 1.3
Different types of fragmentation. (a) Horizontal fragmentation; (b) vertical fragmentation; and (c) hybrid or mixed fragmentation.

1.3 Distributed DB techniques (4)

Different strategies of allocation and replication:

Allocation: each fragment is stored at the site with optimal distribution

Replication: a copy/replica of a fragment is maintained at several different sites.

Strategies for placement of data:

Centralized: Single DB stored at one site with use distributed over the network

Fragmented: Partitions of the db into disjoint fragments, with each fragment assigned to one site

Complete replication: Maintaining a complete copy of DB at each site and is a combination of fragmentation, replication and centralized.

Selective replication: Some data item are fragmented to achieve high locality of reference, and others that are used a many sites and are not frequently updated are replicated and others are centralized.

Distributed transaction processing

Chapter 1 Part 2 Concurrency

1.4 Concurrency control in a Distributed DB (1)

- Aim: The DBMSs control the concurrent execution of user transactions so that the overall correction and update of the database are maintained allowing the user to access the database in a multiprogramming approach safeguarding the misapprehension that each user is working single-handedly on a dedicated system.
- Chaos if not managed:

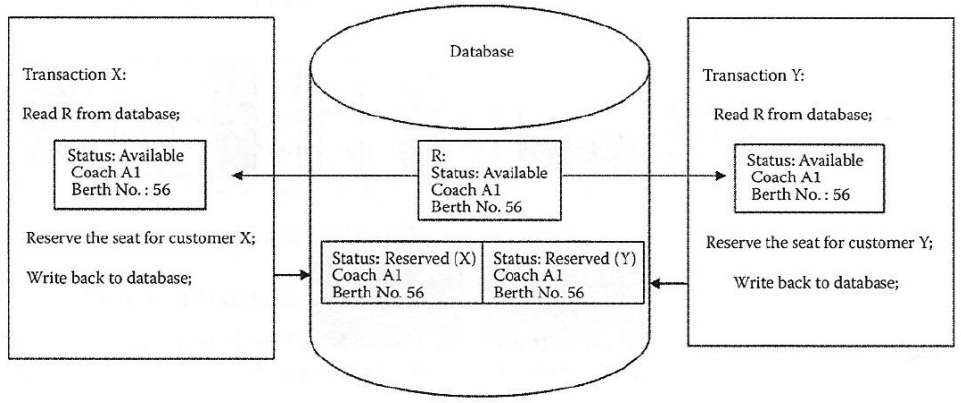


FIGURE 1.4
Scenario of database in the absence of concurrency control module.

1.4 Concurrency control in a Distributed DB (2)

- Centralised DBMSs implement Two-Phase locking (2PL) – to much emphasis on communication in distributed system
- ACID (atomicity, consistency, isolation and durability) must also be maintained in a distributed system and requires: recoverable processes and commit protocol

1.4 Concurrency control in a Distributed DB (3)

Extensions to 2PL to DDBMS:

1a. Centralised 2PL: Single site responsible for lock management (LM) for the whole DDBMS. Coordinating TM (TM at site where transaction is imitated) make can make all locking requests on behalf of the local TMs.

+ Easy to implement

- Bottlenecks and lower reliability;
- replica control protocol is additionally needed if data are replicated.

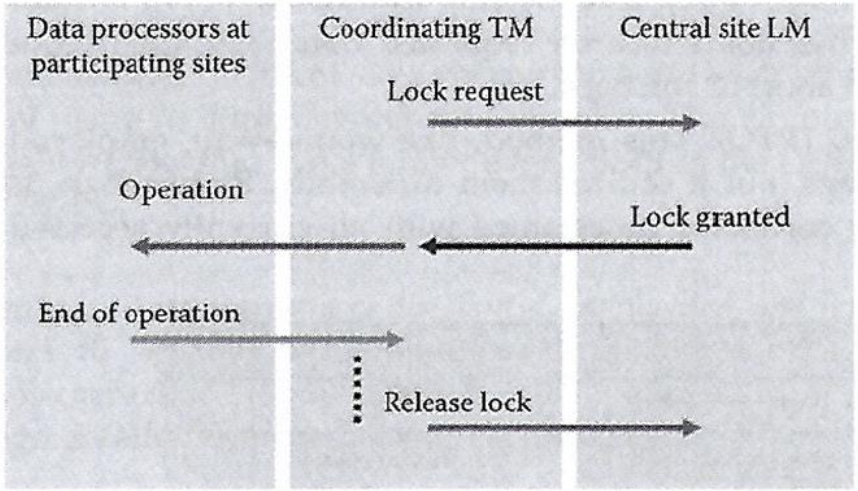


FIGURE 1.5
Communication in centralized 2PL in distributed environment.

1.4 Concurrency control in a Distributed DB (4)

1b. Primary copy 2PL: Several LMs are distributed to a number of sites and each LM is responsible for managing the locks for a set of data items. For replica data items, one copy is chosen as the primary copy, and others are slave copies. Only the primary copy of a data item that is updated needs to be write-locked. Once the primary copy is updated the change is propagated to the slaves:

+ Lower communication costs are better performance than centralised 2PL

- Deadlock handling is more complex in this algorithm

1.4 Concurrency control in a Distributed DB (5)

1c. Distributed 2PL: LMs are distributed to all sites. Each LM is responsible for locks for data at that site. Same as primary 2PL if data is not replicated. ROWA (read one, write all) is implemented if data are replicated.

Read (x): Any copy of a replicated item x can be read by obtaining a read-lock on that copy

Write(x): All copies of x must be write-locked before x can be updated

Communication: The coordinating TM sends the lock request to the LMs of all participating sites, and the LMs pass the operations to the data processors. End of operation is signalled to coordinating TM.

+ Better than primary copy 2PL

- More complex and more communication

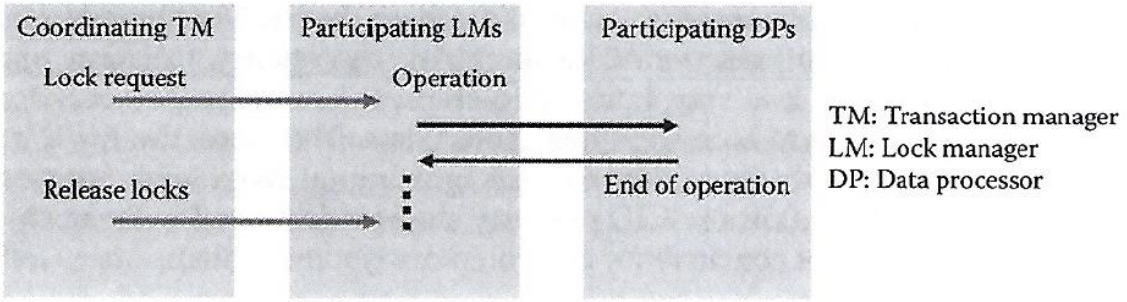


FIGURE 1.6
Communication in distributed 2PL in distributed environment.

1.4 Concurrency control in a Distributed DB (5)

2. Wound-Wait

- Also “Read-any, Write all” strategy. 2PL waits for data, WW avoids deadlocks with timestamps.
- Each transaction is numbered according to its initial start-up time, and younger transactions are prevented from making older ones wait.
- If an older transaction requests a lock, and if the request would direct to the older transaction waiting for a younger transaction, the younger transaction is “wounded” and it is restarted unless it is already in the second phase of its commit protocol. Younger transactions can wait for older transactions.
- + no deadlocks
- See example on p.8

	T1 is allowed to
t(T1) > t(T2) ⟶	Wait
t(T1) < t(T2) ⟶	Abort and rollback

1.4 Concurrency control in a Distributed DB (6)

3. Basic Timestamp Ordering (BTO)

- Use timestamps differently from method of WW.
- Rather than using a locking approach, BTO correlates timestamps with all currently accessed data items and requires that conflicting data accesses by transactions be performed in timestamp order. Transactions that attempt to perform out-of-order accesses are restarted.
- When a read request is received for an item, it is permitted if the timestamp of the requester exceeds the item's write timestamp.
- When a write request is received, it is permitted if the timestamp of the requester exceeds the item's read timestamp.
- If the timestamp is less than the current timestamp. The request is ignored.
- For replicated data: read any, write all is implemented

1.4 Concurrency control in a Distributed DB (7)

4. Distributed optimistic (OPT):

Timestamped based operating by exchanging certification information during the commit protocol

For each data item a read and write timestamp is maintained. Read and writes are done freely and stored in a local workspace until commit time.

When all the transaction's cohorts have completed their work, and have reported back to master, the transaction is assigned a globally unique timestamp. This timestamp is sent to each cohort in the "prepare to commit" message, and is used locally to certified read and writes:

A read is certified if (a) the version that was read is still the current version of the item and (b) no write with a newer timestamp has already been certified.

A write is certified if (a) No later reads have been certified and subsequently committed, and (b) no later reads have been locally certified already

1.5 Promises of DDBMS (1)

DDBMS: makes distribution transparent to user; facilitates quick and easy access of data for users who are located at different locations. Every site is considered as a database system of its own and supervised autonomously, and local data are stored on a local computer db and different sites are connected to each other using high-speed network connections.

DDBMS has complete functionality of DBMS.

1.5 Promises of DDBMS (2)

1. Transparent management if distributed , fragmented and replicated data.
2. Improved reliability and availability through distributed transactions.
3. Improved performance.
4. Higher system extensibility.
5. Sustained performance (correct textbook).

Distributed transaction processing

Chapter 1 Part 3

2.1 Distributed transaction processing (1)

- Distributed transaction: DB transaction in which two or more network hosts are involved. The transaction is composed of several sub-transactions, each running on a different site.
- DDBMS has 5 components: transactions, TM, data manager (DM), network concurrency control scheduler (CCS), and data.
- Each transaction is supervised by a single TM, which manages distributed computation for that transaction.
- Operations: READ(X); WRITE(X, new value); BEGIN and END for blocks.
- DMs manages the stored DB functioning as back-end DB processors.
- In response to commands from transactions, TMs issue commands to DMs specifying stored data items to be read or written.

2.1 Distributed transaction processing (2)

Component in system can take role of client, server or coordinator.

- *Transactional client* only sees transaction through the transaction coordinator (TC invoke services: begin, commit, abort).
- *Transactional Server* registers its participation in in a transaction with the coordinator – implements 2 phase transactional protocol (2 phase commit).
- *Transactional Coordinator* manages transactions;

  Handles begin/commit/send calls;

  Allocated system-wide unique id;

  Different transactions has different coordinators.

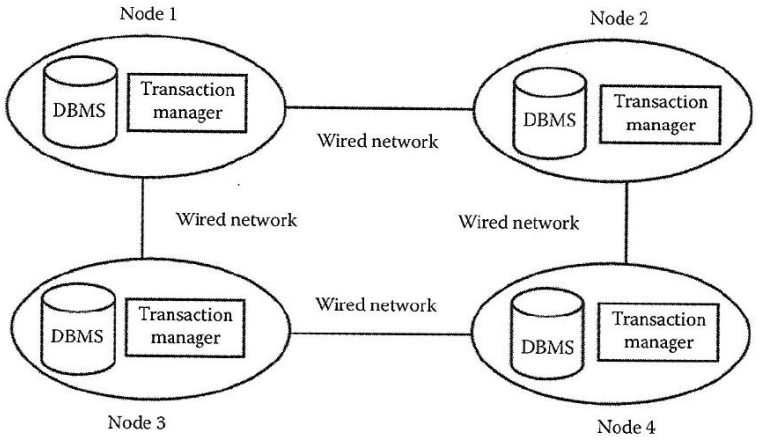


FIGURE 1.8
Transactions in distributed database system.

2.1 Distributed transaction processing (3)

- DM may abort transactions but an atomic commit protocol (ACP) is run by each DM to ensure that all the sub-transactions are consistently committed or aborted.
- ACP ensures:
  - All the DMs reach same decision;
  - Decisions are not reversible;
  - A commit decision can only be reach if all the DMs voted to commit;
  - If there are no failures and all the DMs voted to commit, the discussion will be commit;
  - If all failures are repaired, without any new failures, all the DMs will eventlually reach a decision.

2.2 Distributed transaction processing models (1)

2.1 Atomic actions and flat transactions

- Atomic action: unitary action or object that is essentially indivisible, unchangeable, whole and irreducible.
- Flat transaction consists of atomic actions performed on local variables by accessing single DBMS using call or statement level interface. Indivisible from client perspective.
- Total rollback when aborted, all variables restored to their previous value.
- Entire transaction fails if there is a problem.

2.2 Distributed transaction processing models (2)

2.1 Nested transactions

- Nested transaction: transaction that is initiated by an instruction within the scope of an already started transaction.
- One transaction has many sub-transactions, with their own sub-transactions…. parents has children
- Root transaction is called top-level (TL) transaction;
- Nested transactions enable committing and aborting the sub-transactions independently of the larger transactions.
- Refer to figures 1.9 and 1.10 on pages 12 and 13.

2.2 Distributed transaction processing (DTP) models (3)

Nested transactions has RULES:

1. When the child is active, the parent may not perform any operations other than to commit or abort, or to create more sub-transactions.
2. The commit operation performed on the sub-transactions has no effect on the state of the parent transaction, parent transactions are still uncommitted. The parent can view the modification made by the child transaction but these modifications will be hidden to all other transactions until the parent transaction also commits.
3. If the parent transaction is committed or aborted, the same will happen to the child.
4. The depth of the nested transaction is limited only by memory and cannot be restricted by external mechanisms.

Distributed transaction processing

Chapter 1 Part 4 ACID vs CAP

3. DTP in relational and non-relational DBs (1)

3.1 DTP in Relational DB

- Transactions compliant with ACID (atomic, consistent, isolated and durable)
- DTP takes place as follows:
  1. The *application requester (AR)* accepts requests in the form of SQL Query form application, and sends it to appropriate application server.
  2. The *application server (AS)* processes the section that it can process and forward the remainder to DB servers for subsequent processing, using Application Support Protocol (ASP) which handled data representation conversion.
  3. The *database server (DS)* supports distributed requests and forward parts of the request to collaborating DSs, using Database Support Protocol (DSP)
- *Decomposition of SQL enables optimization: determine the number of tables to be sent through the network!*
- *See example on p15.*

3. DTP in relational and non-relational DBs (2)

3.1 DTP in NON- Relational DB

- Transactions compliant with 2of 3 from CAP (Consistency, Partition tolerance and Availability);
- Non-relational DBs does not incorporate the table/key model- uses key-value pairs in documents; use aggregate data models
- Aggregate: Collection of data (unit) that makes it easier for the DB to handle data storage over group when the unit of data resides on any machine.

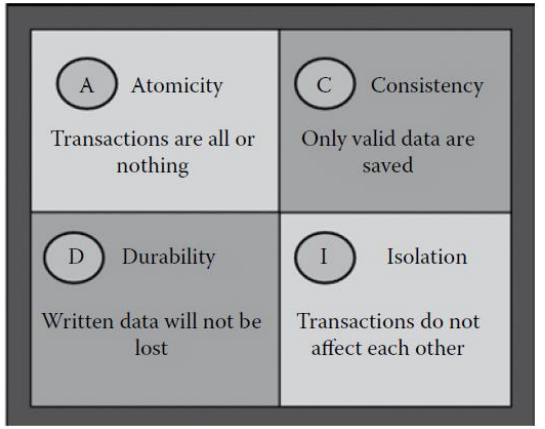
3. DTP in relational and non-relational DBs (3)

Two types of distributing data:

1. *Sharding* distributes data across multiple servers, so each server acts and the single source for a subset of data;
2. *Replication* copies data across multiple servers, so each bit of data can be found at multiple places. 2 ways:
    1. *Maser-slave replication* make on node the authoritative copy, while slaves synchronise with master and may handle reads;
    2. *Peer-to-peer replication* allows writes to any node; the nodes coordinate to synchronise their copies of the data.

3. ACID property in DTP

FIGURE 1.11
Significance of ACID property.



1. ACID revisited:

Atomicity refers to the ability of the DBMS to guarantee that either all of the jobs of a transaction are performed or none of them and database modifications must follow an "all or nothing" rule. If some part of a transaction fails, then the entire transaction fails, and vice versa.

Consistency property ensures that the database remains in a consistent state, despite the transaction succeeding or failing and both before the start of the transaction and after the transaction is over.

Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction and helps to retain concurrency of database.

Durability states that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. That means when users are notified of success, the transactions will persist, not be undone, and survive from system failure.

3.2 ACID Property and Non-Relational Database(1)

- As industries become more and more reliant upon a large quantity of unorganized data such as images, text, files, audio, and videos, traditional RDBMS technologies are proving to be a bottleneck in such kind of situations as they feature a very strict schema.
- NoSQL is a relatively new DBMS technology for handling large amounts of *unstructured data* that does not come in a predefined format.
- Most non-relational databases are incorporated into websites such as Google, Yahoo, Amazon, and Facebook
- The most important feature of a non-relational database is its scalability

3.2 ACID Property and Non-Relational Database (2)

RDBMSs are compliant with the ACID, but NoSQL follows a different approach, that is, CAP theorem laid down by Eric Brewers.

1. *Consistency*: All clients view the same instance of data at the same time.
2. *Availability*: Database can be updated, added, or removed without going offline.
3. *Partition tolerance*: Databases can be distributed across multiple servers, and they are tolerant to network failures.

This theorem states that it is mathematically impossible for an NoSQL DBMS to guarantee all three features. One must always pick two out of the three (C, A, P), that is, CA o r CP or AP. The various databases that come under different categories are:

1. *CA category*: RDBMS (MySQL, Postgres)
2. *CP category*: BigTable, HBase, HyperTable, MongoDB, Terrastore, Scalaris, and Redis.
3. *AP category*: Dynamo, Voldemort, CouchDB, SimpleDB, Riak, Tokyo Cabinet, and Cassandra.

3.2 ACID Property and Non-Relational Database (3)

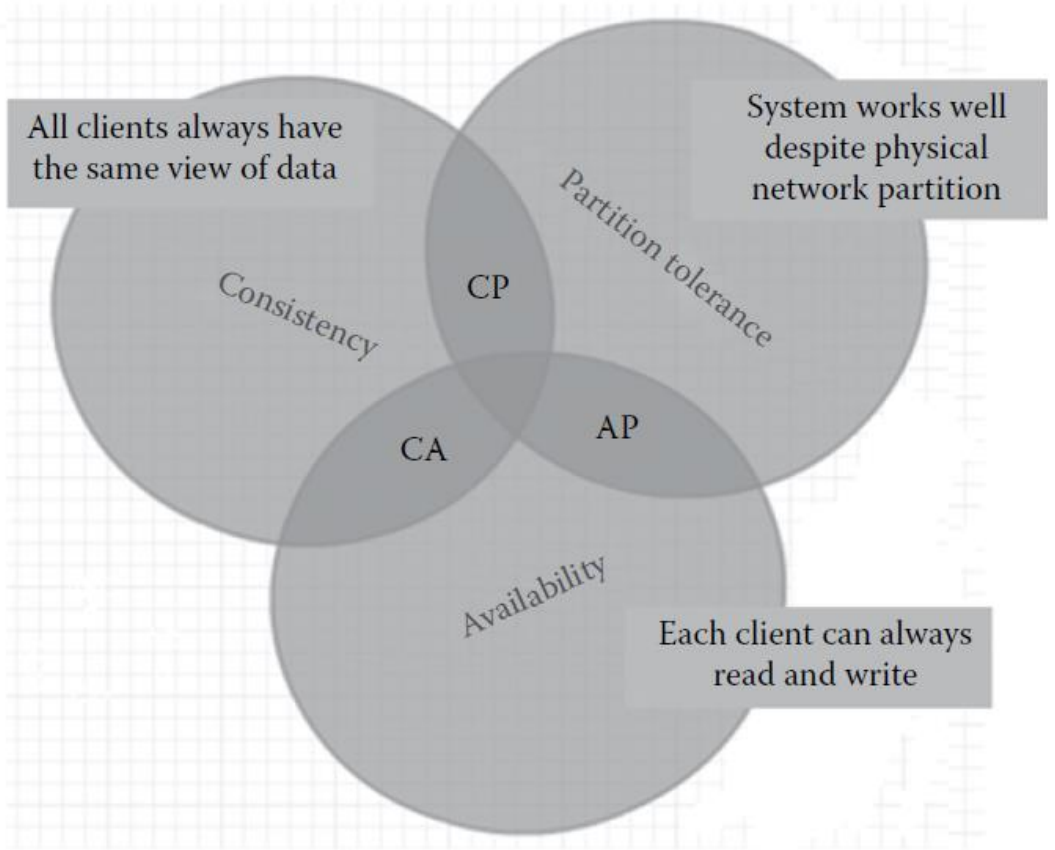


FIGURE 1.12
Significance of CAP theorem.

Distributed transaction processing

Chapter 1 Part 5

4. NoSQL in Distributed Transaction Processing (1)

- NoSQL means Not Only SQL, implying that when designing a software solution or product, there is more than one storage mechanism that could be used based on the needs.
- NoSQL does not have a rigid definition, but NoSQL does not employ relational model, is open source, schema less, and meant for unstructured data storage and retrieval.
- There are four types of NoSQL databases: (a) column family stores, (b) key–value pairs, (c) document store, and (d) graph databases.

4. NoSQL in Distributed Transaction Processing (2)

Column family stores: Column family databases store data in column families as rows that have many columns associated with a row key.

- Column families are groups of related data that are often accessed together. Cassandra is one of the popular column family databases; HBase, Hypertable, and Amazon DynamoDB are others

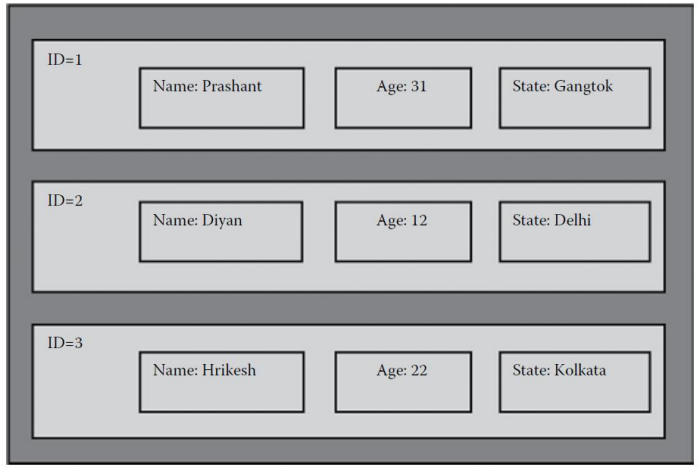


FIGURE 1.13
Column family store database.

4. NoSQL in Distributed Transaction Processing (3)

Key–value pairs: Key–value pair database is one of the simplest NoSQL data stores to use from an Application Programming Interface (API) perspective.

- The client can get the value for the key, put a value for a key, or delete a key from the data store.
- Some of the popular key–value databases are Riak, Redis, Berkeley DB, Amazon DynamoDB, and Couchbase.

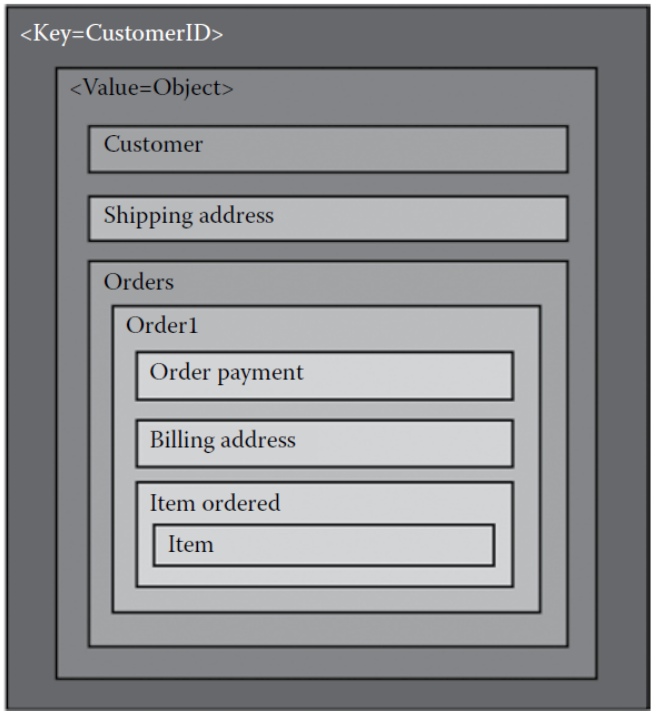


FIGURE 1.14
Key–value pair database.

4. NoSQL in Distributed Transaction Processing (4)

Document store: Documents are the main concept in document databases.

- It stores the data in JSON-like documents having key–value pairs.
- MongoDB is a document database that stores data as a hierarchy of key–value pairs, which allows branching at different levels (a maximum of three levels).
- Some of the popular document databases are MongoDB, CouchDB, Terrastore, OrientDB, RavenDB, and Lotus Notes that use document storage.
- Document database such as MongoDB provide a rich query language and constructs such as database and indexes allowing for easier transition from relational databases.
- Also, MongoDB stores data in JSON-like .BSON files.

```
{
    "First_name": "Prashant",
    "Middle_name": "Neopaney",
    "Last_name": "Chettri",
    "Age": 31,
    "Address: "{
                "Street": "Namthang",
                "City": "Namchi"
                },
}
```

4. NoSQL in Distributed Transaction Processing (5)

Graph databases: These databases store entities and relationships between these entities.

- Entities are also known as nodes that have properties, and relations are known as edges that can have properties.
- There are many graph databases such as Neo4J, Infinite Graph, OrientDB, or FlockDB.

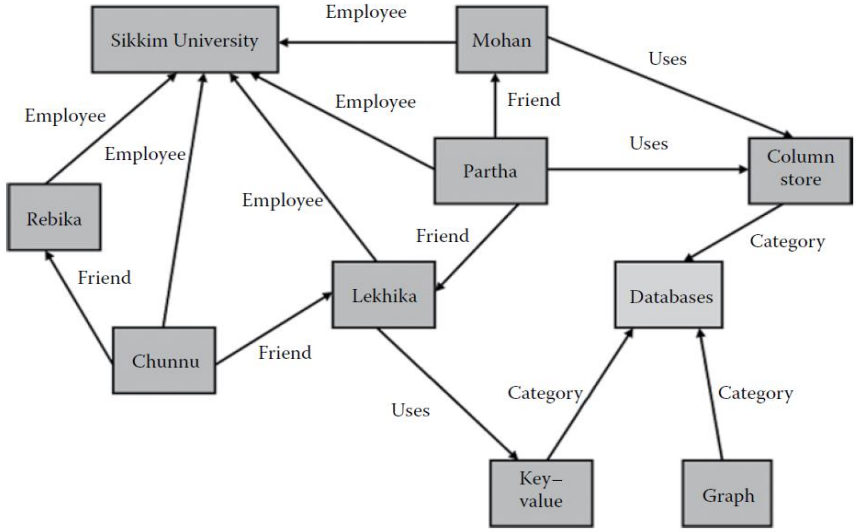


FIGURE 1.15
Graph database.

4. NoSQL in Distributed Transaction Processing (6)

NoSQL has several disadvantages such as the following:

1. No standardized query as that of RDBMS;
2. Lacks ACID compliance, which is required in some situations;
3. No evident increase in performance when used for structured data

MAKE A WISE CHOICE!

4. NoSQL in Distributed Transaction Processing (7)

TABLE 1.2

Comparison of NoSQL and SQL

Basis of Comparison	NoSQL (MongoDB)	SQL (RDBMS)
Data Storage	Stored as key–value pairs in documents.	Stored in a relational model as rows and columns (tables).
Schema Flexibility	Dynamic schema; data can be added, updated, or deleted anytime.	Fixed schema. Altering will result in going offline temporarily.
Specialty	Data which have no definite type or structure.	Data whose type is known in advance.
Scaling	Horizontal: data are stored across multiple servers.	Vertical: more data means bigger servers to handle them.
ACID Compliance	Sacrifice ACID for scalability and performance.	Full compliance with ACID.

4. NoSQL in Distributed Transaction Processing (8)

TABLE 1.3

Differences in Queries between NoSQL and SQL

Syntax/Query	NoSQL	SQL
To create database	`Use db_name`	`CREATE DATABASE db_name`
To drop database	`db.dropDatabase()`	`DROP DATABASE db_name`
To insert data	`db.db_name.insert ({title : "NoSQL"})`	`INSERT INTO table_name VALUES ("SQL")`
To update data	`db.db_name.update ({'title': 'NoSQL'}, {$set:{'title':'NewNoSQL'}})`	`UPDATE table_name SET title = 'NewNoSQL'`

5. Security Issues in DTP Systems (1)

- A security constraint consists of a data specification (any subset of transaction processing system) and a security value (given by a classification function). The specific values are unclassified, confidential, secret, and top secret
- Two types of security constraints: internal and external constraints.
    1. *Internal constraints classify* the entire Transaction Processing System (TPS) as well as relations, attributes, and tuples within a relation. These constraints can be applied to data as they are actually stored in the TPS.
    2. *External constraints classify* relationships between data and the results obtained by applying operations on the stored data, such as sum, average, and count. Among these constraints are the functional constraints and the dynamic constraints. These security constraints are subject to inconsistency and conflicting local security constraints. A good global security approach should reject inconsistent security constraints and inconsistent clearance of users.

5. Security Issues in DTP Systems (2)

Examples of the *inconsistencies*:

1. *Conflicting security constraints:* such constraints classify the same facts into different categories;
2. *Overlapped security constraints:* these constraints cover overlapped data domains;
3. *Inconsistent security level of replicated data:* cases where different copies of replicated data may belong to different security cases;
4. *Access privileges of users to replicated data:* instances where a user may have different access rights on replicated data at different sites.